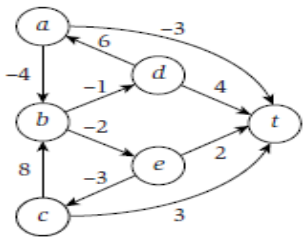
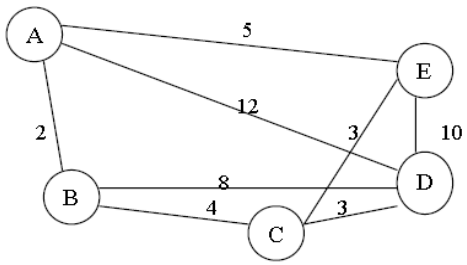
RAMAIAH Institute of Technology

Department of Computer Science & Engineering

QUESTION BANK FOR IV SEMESTER

<table>
<tr><th colspan="2">Lab Programs</th><th>CO</th><th>PO</th></tr>
<tr><td>1.</td><td>A drama venue needs to be allocated for different drama school requests such that maximum profit is obtained for the company owning the drama venue. The requests are shown in the table with start–time, finish-time and the amount affordable by the drama school. Design and implement Weighted Interval Scheduling algorithm such that maximum profit is obtained for the company owning the drama venue using Dynamic programming principles. State the design strategy used and comment on the time complexity of the same.
<table>
<tr><th>Drama School</th><th>Start-time</th><th>Finish-time</th><th>Value</th></tr>
<tr><td>1</td><td>1</td><td>2</td><td>100</td></tr>
<tr><td>2</td><td>2</td><td>5</td><td>200</td></tr>
<tr><td>3</td><td>3</td><td>6</td><td>300</td></tr>
<tr><td>4</td><td>4</td><td>8</td><td>400</td></tr>
<tr><td>5</td><td>5</td><td>9</td><td>500</td></tr>
<tr><td>6</td><td>6</td><td>10</td><td>100</td></tr>
</table></td><td>2</td><td>1,2,3,4,5,12</td></tr>
<tr><td>2.</td><td>Given a set of non-negative integers and a value of variable sum. design and implement an algorithm to determine if there is a subset of the given set with a sum equal to the given sum. A suitable message is to be displayed if the given problem instance doesn't have a solution.</td><td>2</td><td>1,2,3,4,5,12</td></tr>
<tr><td>3.</td><td>Alia is planning for a trekking expedition with a backpack that can hold 7kg. She needs to select the most valuable items from the following list that can be accommodated within the backpack. Design and implement Knapsack algorithm that displays the most valuable items that can be carried by her using Dynamic programming principle and find the time complexity of the same.
<table>
<tr><th>Items</th><th>Weight</th><th>Value</th></tr>
<tr><td>1</td><td>3</td><td>10</td></tr>
<tr><td>2</td><td>5</td><td>4</td></tr>
<tr><td>3</td><td>6</td><td>9</td></tr>
<tr><td>4</td><td>2</td><td>11</td></tr>
</table></td><td>2</td><td>1,2,3,4,5,12</td></tr>
<tr><td>4.</td><td>Design and implement Bellman ford algorithm to find the shortest path from a given source to all other nodes. State the design strategy used and comment on the time complexity of the same.

</td><td>2</td><td>1,2,3,4,5,12</td></tr>
</table>

5.	Design and implement **N-queens algorithm** that displays the possible solutions on 4 x 4 chessboard. State the design strategy used and time complexity of the same.	3	1,2,3,4,5,12
6.	Design and implement an algorithm for Travelling salesman problem using backtracking technique. 	3	1,2,3,4,5,12

Marks Distribution:

Write-Up	Algorithm Implementation and Execution	Viva	Change of Program
8 Marks	**35 Marks**	**7 Marks**	**-5 Marks**

Q 1.

```
#include <stdio.h>
#include <stdlib.h>
typedef struct { int start, end, value; } Interval;
int cmp(const void *a, const void *b) {
 return ((Interval*)a)->end - ((Interval*)b)->end;
}
int binarySearch(Interval intervals[], int index) {
 int lo = 0, hi = index - 1;
 while (lo <= hi) {
 int mid = (lo + hi) / 2;
 if (intervals[mid].end <= intervals[index].start) {
 if (intervals[mid + 1].end <= intervals[index].start) lo = mid
+ 1;
 else return mid;
 } else hi = mid - 1;
 }
```

```
 return -1;
}
int weightedIntervalScheduling(Interval intervals[], int n) {
 qsort(intervals, n, sizeof(Interval), cmp);
 int dp[n + 1];
 dp[0] = 0;
 for (int i = 1; i <= n; i++) {
 int inclProfit = intervals[i-1].value;
 int l = binarySearch(intervals, i-1);
 if (l != -1) inclProfit += dp[l+1];
 dp[i] = (inclProfit > dp[i-1]) ? inclProfit : dp[i-1];
 }
 return dp[n];
}
int main() {
 Interval intervals[] = {{1,2,100}, {2,5,200}, {3,6,300},
{4,8,400}, {5,9,500}, {6,10,100}};
 int n = sizeof(intervals) / sizeof(intervals[0]);
 printf("Maximum profit: %d\n",
weightedIntervalScheduling(intervals, n));
 return 0;
}
```

Q2.

```
#include <stdio.h>
#include <stdbool.h>
// Function to check if there is a subset with the given sum
bool isSubsetSum(int set[], int n, int sum) {
 bool dp[sum + 1];
 dp[0] = true;
 for (int i = 1; i <= sum; i++)
 dp[i] = false;
 for (int i = 0; i < n; i++)
```

```
 for (int j = sum; j >= set[i]; j--)
 if (dp[j - set[i]])
 dp[j] = true;
 return dp[sum];
}
int main() {
 int set[] = {3, 34, 4, 12, 5, 2};
 int sum = 9;
 int n = sizeof(set) / sizeof(set[0]);
 if (isSubsetSum(set, n, sum))
 printf("There is a subset with sum %d\n", sum);
 else
 printf("No subset with sum %d\n", sum);
 return 0;
}
```

Q3.

```
#include <stdio.h>
#include <stdlib.h>
#define MAX(a, b) ((a) > (b) ? (a) : (b))
int knapsack(int W, int wt[], int val[], int n) {
 int i, w, K[n+1][W+1];
 for (i = 0; i <= n; i++) {
 for (w = 0; w <= W; w++) {
 if (i == 0 || w == 0)
 K[i][w] = 0;
 else if (wt[i-1] <= w)
 K[i][w] = MAX(val[i-1] + K[i-1][w-wt[i-1]], K[i-1][w]);
 else
 K[i][w] = K[i-1][w];
 }
 }
```

```
 // Backtrack to find items
 int res = K[n][W];
 w = W;
 for (i = n; i > 0 && res > 0; i--) {
 if (res != K[i-1][w]) {
 printf("Item %d is included\n", i);
 res -= val[i-1];
 w -= wt[i-1];
 }
 }

 return K[n][W];
}
int main() {
 int val[] = {10, 4, 9, 11};
 int wt[] = {3, 5, 6, 2};
 int W = 7;
 int n = sizeof(val) / sizeof(val[0]);
 printf("Maximum value: %d\n", knapsack(W, wt, val, n));
 return 0;

}
```

Q4.

```
#include <stdio.h>
#include <stdlib.h>
#include <limits.h>

#define V 6 // Number of vertices
#define E 9 // Number of edges
```

```
// Structure to represent an edge
struct Edge {
    int src, dest, weight;
};

// Function to implement Bellman-Ford algorithm
void BellmanFord(struct Edge* edges, int source) {
    int dist[V];
    int i, j;

    // Initialize distances
    for (i = 0; i < V; i++)
        dist[i] = INT_MAX;
    dist[source] = 0;

    // Relax all edges V-1 times
    for (i = 1; i <= V - 1; i++) {
        for (j = 0; j < E; j++) {
            int u = edges[j].src;
            int v = edges[j].dest;
            int weight = edges[j].weight;
            if (dist[u] != INT_MAX && dist[u] + weight <
dist[v])
                dist[v] = dist[u] + weight;
        }
    }

    // Check for negative-weight cycles
    for (i = 0; i < E; i++) {
        int u = edges[i].src;
        int v = edges[i].dest;
        int weight = edges[i].weight;
        if (dist[u] != INT_MAX && dist[u] + weight < dist[v]) {
            printf("Graph contains negative weight cycle\n");
            return;
```

```
        }
    }

    // Print the distances
    printf("Vertex   Distance from Source\n");
    for (i = 0; i < V; i++)
        printf("%d \t\t %d\n", i, dist[i]);
}

int main() {
    struct Edge edges[E] = {
        {0, 1, -4}, {0, 3, -3}, {1, 2, -2},
        {1, 3, 4},  {2, 4, 2},  {2, 5, 1},
        {3, 1, 1},  {3, 4, 5},  {4, 5, -3}
    };

    BellmanFord(edges, 0);  // 0 is the source vertex

    return 0;
}
```

Q5.

```
#include <stdio.h>
#include <stdbool.h>
#define N 4
void printSolution(int board[N][N]) {
 static int count = 1;
 printf("Solution %d:\n", count++);
 for (int i = 0; i < N; i++) {
 for (int j = 0; j < N; j++)
 printf("%c ", board[i][j] ? 'Q' : '.');
```

```
 printf("\n");
 }
 printf("\n");
}
bool isSafe(int board[N][N], int row, int col) {
 for (int i = 0; i < col; i++)
 if (board[row][i])
 return false;
 for (int i = row, j = col; i >= 0 && j >= 0; i--, j--)
 if (board[i][j])
 return false;
 for (int i = row, j = col; j >= 0 && i < N; i++, j--)
 if (board[i][j])
 return false;
 return true;
}
bool solveNQUtil(int board[N][N], int col) {
 if (col >= N) {
 printSolution(board);
 return true;
 }
 bool res = false;
 for (int i = 0; i < N; i++) {
 if (isSafe(board, i, col)) {
 board[i][col] = 1;
 res = solveNQUtil(board, col + 1) || res;
 board[i][col] = 0;
 }
 }
 return res;
}
void solveNQ() {
 int board[N][N] = {0};
 if (!solveNQUtil(board, 0))
 printf("Solution does not exist");
```

```
}
int main() {
 solveNQ();
 return 0;

}
```

Q6.

```
#include <stdio.h>
#include <limits.h>
#define V 5 // Number of vertices
int graph[V][V] = {
 {0, 2, 0, 12, 5},
 {2, 0, 4, 8, 0},
 {0, 4, 0, 3, 3},
 {12, 8, 3, 0, 10},
 {5, 0, 3, 10, 0}
};
int visited[V];
int path[V];
int min_cost = INT_MAX;
int final_path[V];
void copy_path() {
 for (int i = 0; i < V; i++)
 final_path[i] = path[i];
 final_path[V] = path[0];
}
void tsp(int curr, int count, int cost) {
 if (count == V && graph[curr][0]) {
 if (cost + graph[curr][0] < min_cost) {
 min_cost = cost + graph[curr][0];
 copy_path();
 }
```

```
 return;
 }
 for (int i = 0; i < V; i++) {
 if (!visited[i] && graph[curr][i]) {
 visited[i] = 1;
 path[count] = i;
 tsp(i, count + 1, cost + graph[curr][i]);
 visited[i] = 0;
 }
 }
}
int main() {
 visited[0] = 1;
 path[0] = 0;
 tsp(0, 1, 0);
 printf("Minimum cost: %d\n", min_cost);
 printf("Path: ");
 for (int i = 0; i <= V; i++)
 printf("%c ", 'A' + final_path[i]);
 printf("\n");
 return 0;

}
```